Lecture 2 - C++ Basics (Cont.)

"Fifty years of programming language research, and we end up with C++?" - Richard A. O'Keefe

- Pass by *value*, *reference* and *address*
- Stack / heap
- Smart pointers
- Classes

Last Week Recap

- C++ Basics

Memory

We can view the memory structure as a 1-dimensional array of bytes:
`[0,0,..,0]`

Program Memory

The OS allocates a program some RAM to hold data and code and keep track of the execution.

- **Heap**: a block of memory that we can allocate for use
- **Stack**: the LIFO structure in which local variables are stored

Managed vs unmanaged languages

- Managed languages like Java allow the programmer to focus on the task at hand and forget about memory
- But, difficult to talk to hardware without explicit control of memory structure
- Unmanaged languages like C++ give the programmer total control over the memory allocated to your program

Allocating Memory

On the stack:

```
int i = 0;
Point2 p(0, 0);
```

On the heap (note, raw pointer):

```
int * i = new int;
Point2 * p = new Point2(0, 0);
```

Activity - Space Invaders / Breakout code

Let's examine some memory allocations in Java and in C++.

Recall Functions

Small chunks of a program (methods in Java):

```
int add(int x, int y) {
    return x + y;
}
```

- how to pass variables in and out of a function, and
- the scope of variables in a function.

Passing variables

- Pass by value (copy): when calling a function we pass a copy of our variable to it. |
- Pass by reference (not Java reference!): when calling a function we pass the variable itself, possibly with an alternative name, a so-called reference. |
- Pass by address (pointer): when calling a function we pass the memory address of our variable. |

Pass by value

```
int add(int x, int y) {
    return x + y;
}

int a = 2;
int b = 4;

add(a, b);
```

- Arguments are copied into the function, hence `a` and `b` cannot be modified.
- Copying can affect performance but passing primitives is fine

Pass by reference

```
int add(int& x, int& y) {
    return x + y;
}

int a = 2;
int b = 4;

add(a, b);
```

- `x` and `y` are treated the same as `a` and `b`, so can be modified if needed.
- No copying
- At the call site there is no difference between pass by *value* and *reference*.

Pass by address

```
int add(int* x, int* y) {
    return *x + *y;
}

int a = 2;
int b = 4;

add(&a, &b);
```

- Values of `a` and `b` can be modified if needed.
- No copying
- `nullptr` checks, i.e. the pointer may not point to anything.

Activity

Find some code in the assignment codebase that passes arguments:

1. by copy
2. by reference
3. by raw pointer

Returning values

As if the above wasn't enough ...

Values can also be returned from a function by value, reference or address.

```
int add() {
}
int& add() {
}
int* add() {
}
```

Managing heap

If you allocated a memory on the heap you must delete it!

```
int * i = new int;

// do something useful with i

delete i;
i = nullptr;
```

Java vs C++ (NOT equivalent!)

Java garbage collection (GC) is not the same as memory deallocation in C++.

```
Point2 p = new Point2(0, 0);
p = null;
Point2 * p = new Point2(0, 0);
delete p;
p = nullptr;
```

Summary

- Memory management in C++ is often considered to be tricky
- There is a better way (smart pointers)
- Know your stack and heap
- Also know pass by value, reference and address

Why Pointers?

We can't use a reference to store something dynamically allocated and copying can be costly in terms of memory and performance.

Trouble with Pointers

What is the problem here?

```
int someFunction() {
    int * i = new int;
    return 0;
}
int* someFunction() {
    int a = 5;
    return &a;
}
```

Use Smart Pointers

```
#include <memory>
#include <string>

int main() {
    std::shared_ptr<std::string> s = std::make_shared<std::
    return 0;
}
```

A bit too verbose!

Use auto

```
#include <memory>
#include <string>

int main() {
    auto s = std::make_shared<std::string>("Hello World");
    return 0;
}
```

Use namespace

Not always recommended, but OK for our simple programs.

```
#include <memory>
#include <string>

using namespace std;

int main() {
    auto s = make_shared<string>("Hello World");
    return 0;
}
```

Dealing with Legacy Code

- Many C/C++ programs still use (sometimes necessarily) raw pointers (*)
- Possible to write a “wrapper” around legacy code

Demo Time

Assignment code uses smart pointers, let's examine them.

Summary

- Smart pointers are *smart* :)
- They simplify the object life-cycle – less worrying about memory leaks

Case Study (Activity)

Design and implement a scoreboard.

Classes (Semantics)

A class allows us to abstract models and encapsulate data.

Classes (C++ Syntax)

C++ classes have two parts:

- Header file: `Scoreboard.h` (the *what*)
- Implementation file: `Scoreboard.cpp` (the *how*)

In Java this would just be `Scoreboard.java`

Designing a Class (an API)

A scoreboard has a score. What is `score`?
Is it a piece of text (“hello”), an integer (3), a real number (5.14) or some other type?

Implications of Class Design

- If we use an integer then we can't use, say 0.5. Is there a game that uses fractions for score?
- If we use a real number then you might run into rounding errors.

Header

Scoreboard.h

```
class Scoreboard {
    public:
        Scoreboard();
        void increment(int x);
};
```

Is this good API? Why yes / no?

Implementation

Scoreboard.cpp

```
#include "Scoreboard.h"

Scoreboard::Scoreboard() {
    // constructor like in Java
}

void Scoreboard::increment(int x) {
    // a member function, like a Java method
}
```

Use Scoreboard

The `main.cpp` file:

```
#include <iostream>
#include "Scoreboard.h"

int main() {
    Scoreboard s();
    std::cout << "Created a scoreboard." << std::endl;
    return 0;
}
```

Activity

What is missing from Scoreboard functionality?
Have a go at implementing this missing functionality.

Demo Time

Assignment code: let's check out custom classes.

Conclusion

- The syntax of C++ classes differs from Java
- The semantics of C++ classes is very similar to Java
- Using classes is reasonably straightforward

Tutorial

Tutorial link